IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant(s)	:	Shepherd, et al.
Serial No.	:	10/791,019
Filed	:	March 2, 2004
For	:	SECURE BROWSER
Examiner	:	Haoshian Shih
Art Unit	:	2173
Customer No.	:	10037

October 5, 2009

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

APPEAL BRIEF UNDER 37 C.F.R. § 41.37

In response to the Final Office Action dated April 2, 2009, the Notice of Appeal dated August 3, 2009, and the Notice of Panel Decision from Pre-Appeal Brief Review, Applicants provide herewith their Appeal Brief.

(i) Real party in interest.

The real party in interest is Question Mark Computing Limited. An assignment is recorded at Reel 015054, frame 0472.

(ii) Related appeals and interferences.

None.

(iii) Status of claims.

Claims 1-21 are in the application.

Claims 1-21 are rejected.

The rejection of claims 1-21 is appealed.

(iv) Status of amendments.

The amendment after final rejection is entered on appeal. See Advisory Action dated July 20, 2009. However, no formal amendment was submitted for entry.

(v) Summary of claimed subject matter.

Claim 1 provides a secure user interface method, for interacting with a user through a browser (IE in Fig. 2), comprising:

controlling the browser (Workstation in Fig. 1) to request a document ("secure content" in Figs. 1 and 2), from a cooperative server (Server in Fig. 1), the browser providing data export support functionality;

receiving data with the browser in response to the request ("Server responds" in Fig. 1);

automatically determining, based on a received data encoding type, whether a secure browser ("secure browser" in Fig. 1) or a normal browser is to be employed (QMSB MIME type), the secure browser having a set of functionality restricted with respect to the normal browser, to enhance security of a received document against data export (page 9, line 25-page 10, line 2);

receiving the secure content for presentation in the secure browser (Page 5, lines 5-19; Page 11, line 31-page 12, line 12; Fig. 1, "server delivers secure content", Fig. 2, "QSB delivers secure content")); and

communicating an input from the user, through the secure browser, to a cooperative server (Page 4, line 11-page 5, line 19).

Claim 9 provides a secure user interface method, for interacting with a user through a browser (IE in Fig. 2), the browser providing a set of navigational functionality, comprising:

requesting a document ("secure content" in Figs. 1 and 2) from a cooperative server (Server in Fig. 1);

receiving data in response to the request ("Server returns web page" in Fig. 1);

automatically determining, based on a received data type encoding (MIME type of QMSB in Fig. 1), whether a secure browser ("Secure Browser" in Figs. 1 and 2) is required to be employed by a content provider (Server in Fig. 1, "Authenticate" in Fig. 2) or whether an insecure browser is to be employed ("Original Browser" in Fig. 2), the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser (page 9, line 25-page 10, line 2);

invoking the secure browser ("QSB called" in Fig. 2);

receiving the secure content for presentation in the secure browser ("Server delivers secure content" in Fig. 1, "deliver secure content" in Fig. 2); and

communicating an input from the user, through the secure browser, to a cooperative server (Page 4, line 11-page 5, line 19).

(vi) Grounds of rejection to be reviewed on appeal.

Claims 1-21 are rejected under 35 U.S.C. § 112, first paragraph for allegedly failing to comply with the written description requirement.

Claims 1-4 and 6-20 are rejected as being anticipated under 35 U.S.C. § 102(e) over Winneg et al., US 7,069,586.

Claims 5 and 21 are rejected as being obvious under 35 U.S.C. § 103(a) over Winneg et al., US 7,069,586 in view of Chang et al., US 2002/0097416.

(vii) Argument.

REJECTION OF CLAIMS 1-21 UNDER 35 U.S.C. § 112. FIRST PARAGRAPH

Claims 1-21 are rejected under 35 U.S.C. § 112, first paragraph, as allegedly failing to comply with the written description requirement, and in particular, the use of the phrase "or a normal browser is to be employed" in claim 1, and "or whether an insecure browser is to be employed" in claim 9 are asserted by the examiner as failing to be supported by an adequate written description in the specification.

In the application as originally filed, claims 1 and 9 stated:

> 1. A secure user interface method, for interacting with a user through a browser, comprising:
>
> controlling the browser to request a document from a cooperative server, the browser providing data export support functionality;
>
> receiving data with the browser in response to the request;
>
> automatically determining, based on a type encoding of the received data, whether a secure browser is to be employed, the secure browser having a set of functionality restricted with respect to a normal browser, to enhance security of a received document against data export;
>
> receiving the secure content for presentation in the secure browser; and
>
> communicating an input from the user, through the secure browser, to a cooperative server.
>
> 9. A secure user interface method, for interacting with a user through a browser, the browser providing a set of navigational functionality, comprising:
>
> requesting a document from a cooperative server;
>
> receiving data in response to the request;
>
> automatically determining whether a secure browser is required to be employed by a content provider, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser;

invoking the secure browser;

receiving the secure content for presentation in the secure browser; and

communicating an input from the user, through the secure browser, to a cooperative server.

As such, these claims alone provide "written description" of the invention, and antecedent basis for the claim language now employed.

Figs. 1 and 2 also clearly show the process for invocation of a secure browser, and the "data encoding type" or "data type encoding" as a basis for selecting the secure browser, as opposed to the alternate, a normal browser (per claim 1) or an insecure browser (per claim 9). See especially Fig. 2, "Content includes triggering link to secure content" and "QSB delivers secure content".

The specification includes an Appendix which includes operative code which implements various portions of the method, demonstrating authentication of the secure browser and launch of the secure browser (an error message of the launch is failed).

As basic "written description" for a "normal browser", "secure browser" and an "insecure browser" is found in at least the following passages, which state:

Page 1, line 8:

BACKGROUND OF THE INVENTION

Secure browsers are designed to provide a secure environment to deliver valuable content and assessments such as tests and exams. Web servers can deliver questions to any web browser, but most browsers are designed to be as open and flexible as possible. When you are delivering secure content or assessments online you need far more security than most browsers provide.

With a secure browser, a content provider can specify that secure content such as a test or an exam may only be delivered in such manner as to significantly reduce the likelihood of cheating, or inappropriate disclosure of sensitive content. Secure browsers allow a content provider to prevent users from printing questions, using the right-click on the mouse, saving the HTML, viewing the source, and accidentally exiting an assessment in a proctored environment. The look and feel of the screen displayed may otherwise correspond to that of a normal browser, except pages may not be stored (cached) in the history, and certain menu options and icons are not displayed or are made unavailable.

Web browsers are typically flexible and open programs which aid the user in navigating the Internet, running programs or applets, and giving the user full control over what he/she is doing. But when browsers are used to take assessments, it's desirable that the user should not have full control and open access. Since the assessment is designed to measure knowledge or a skill, and sometimes has consequences for passing or failing, it's desirable that what the user can do is restricted; essentially the user should only take the assessment and not be able to perform other tasks. For example, it can be desirable that users should not be able to navigate the Internet (where they might find right answers), communicate with others, run other programs, print the screen or copy the questions to other people and so on.

This need has given rise to "secure browsers" or "locked down browsers" or "kiosk software", which are versions of standard browsers which limit the functions that the user can perform. Computers which are used to deliver assessments therefore typically have secure browsers installed, and these lock down the computers to prevent unauthorized actions while taking an assessment.

These secure browsers fulfill needs in situations where users take assessments on their own. But very often assessments are mixed in with other uses of the computer. For example, a learning management system might accept a student's login and allow him or her to choose an assessment; a student might undertake some online course (where they are allowed free use of their browser) followed by an assessment (where they are not); or a corporate executive might use their corporation's intranet, and then be scheduled for a business rules or product knowledge or safety regulation exam. Therefore, other secure browser products such as the Vantage Vanguard^{TM} 3.0 secure desktop environment, Questionmark's own, prior Perception Secure Browser product, or Software Secure's Securexam Browser lack this flexibility, making full use of the computer in both secure and insecure modes difficult. Other secure browsers need to be specifically launched to take the assessment; they cannot be launched on demand by an ordinary browser, when secure content delivery is required.

In such mixed scenarios, it would be desirable to have a browser which can become secure when an assessment (or other secure content) is started and then become open again when an assessment (or the secure content) is finished.

Essentially the problem may be stated that it is desired that secure content to be called from insecure content, with the secure content run securely. Likewise, it is desired that an open user environment be triggered into a restricted user environment, with some assurance that the restricted conditions be maintained.

See also, p. 5, line 26:

The look and feel of the screen displayed may be very similar to that of a normal browser, such as Internet Explorer, although the pages are not stored in Internet Explorer's history listing, and some navigation buttons and toolbars are usually omitted. Likewise, various components of a host browser or operating system may be employed for content presentation, rendering, and use.

The method of the invention is also clearly described in the specification. For example,

Page 10, line 11:

Launch of a secure browser from a regular browser
Internet technologies allow a MIME type to be specified to indicate to the computer operating system which program should be used to display the content. The MIME type may be defined by the file extension or the Content-type header returned by a web server. The present invention, for example, specifies a new MIME type of "Questionmark Secure Browser" (or equivalent) which starts the secure browser to display the assessment or e-learning content that requires more security than a normal browser would provide.

A web page contains a link (triggering link) that, when accessed, passes a MIME type to indicate that a secure browser is required to display this content.

Indicating which web page to 'get'
While the MIME type specifies that the content must be displayed in a secure browser, it doesn't specify where the content is located. There are three general alternatives: (1) Allow the original link to specify the URL or the content so that the secure browser can call the content using a normal http GET command; (2) Allow the secure browser to have a system configuration that allows the secure browser to be triggered to call a specific URL or IP address; and (3) Allow the secure browser to have a system configuration that allows the secure browser to be trigged to call a specific URL or IP address along with a parameter that was provided as part of the trigger (combining (1) and (2), above – the server URL/IP address is configured within the secure browser while the specific assessment and other details is defined in the trigger). It's also possible to have the cooperating server provide some of the details.

Page 11, line 25:

It is therefore an object of the invention to provide a secure user interface method, for interacting with a user through a browser, the browser providing a set of navigational functionality, comprising requesting a document from a cooperative server; receiving data in response to the request; automatically determining whether a secure browser is required to be employed, for example based on a type code or type encoding, the secure browser defining a set of functionality restricted with respect to the functionality of the browser alone; invoking the secure browser; receiving the secure content for presentation in the secure browser; and communicating an input from the user, through the secure browser, to a cooperative server. The functionality may be limited navigational functionality (e.g., access of unrestricted documents, documents outside a specified set, or access of other applications or windows), data manipulation functionality, data export functionality (e.g., print, copy, save, cut, paste, etc.), or the like. The server may authenticate the secure browser, and likewise, the secure browser may authenticate the server, before presenting the secure content. The secure browser may restrict termination of its own execution.

Page 13, line 8:

According to the present invention, the secure browser may still use basic internet technologies (http, html, MIME types, etc) to ease deployment issues; launch a secure browser from a normal browser; be initiated by a triggering link; and employ secure browser calls back to gather actual content.

Example 1

According to a first embodiment of the invention, a communication steam is provided between a user's computer and a server, exemplified as follows by sample HTTP headers. This interaction is represented in Fig. 1.

Headers sent when browser calls web server
```
GET /q/open.dll HTTP/1.0
Accept: */*
Accept-Language: en-gb
Pragma: no-cache
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
NT 5.0; .NET CLR 1.0.3705)
Host: localhost
```

Headers sent when web server responds to browser
```
HTTP/1.1 200 OK
Server: Microsoft-IIS/5.0
Date: Wed, 14 May 2003 15:03:58 GMT
Content-Type: text/html
```

Secure browser headers
Possible example headers sent when web server responds to browser calling the 'trigger'; note MIME type qmsb.

```
HTTP/1.1 200 OK
```

```
Server: Microsoft-IIS/5.0
Date: Wed, 14 May 2003 15:03:58 GMT
Content-Type: application/qmsb
```

Possible example headers sent by secure browser when calling server; note security code which is passed from secure browser to web server:

```
GET /q/open.dll HTTP/1.0
Accept: */*
Accept-Language: en-gb
Pragma: no-cache
User-Agent: Secure Browser
Security-Code: 47bce5c74f589f4867dbd57e9ca9f808
Security-Expires:15:05:2003:03:25
Host: localhost
```

Page 15, line 14:

The URL to QSBlaunch is, for example, www.xyz.com/qsblaunch.asp and the URL to QSBcheck is www.xyz.com/qsbcheck.asp.

Triggering link
The triggering link consists of the URL to QSBlaunch, followed by arbitrary parameters. The parameters will typically define the assessment to be run. All parameters must be URL encoded in the usual way.

Example triggering links:
www.xyz.com/qsblaunch.asp?Assessment=12345&Group=Potato
www.xyz.com/qsblaunch.asp?Token=12345678901234

What QSBlaunch does

- QSBlaunch does the following:
- It receives on the command line the parameters
- It knows the URL of the QSBcheck.asp or similar program.
- It calls QS Server technologies (QSBst.dll) to pass the url and parameters and gets back a checksum.
- It then constructs a dynamic .qmsb document containing the URL, parameters, version and checksum.
- This .qmsb document is itself encrypted to ensure that QS only will run from an approved source
- It sends it back to the browser with content type .qmsb. ASP code to do this is as follows: Response.ContentType = "application/x-qmsb"
  Response.AddHeader "Content-Disposition", "filename=qsblaunch.qmsb"

The QMSB file will contain a message at the top in case it is read in error, for example,:

"If you are reading this file, then there has been an error in installing the Questionmark Secure. You need to install Questionmark Secure from <url> and you need to set up your browser to call Questionmark Secure on the MIME type of application/x-qmsb."

What happens at the client end after QSBlaunch runs
QSBlaunch sends back a document, which is of MIME type, associated with QSB.

If Questionmark Secure Browser is not installed, there is no way that the participant can proceed. They have to install Questionmark Secure Browser and re-run the triggering link. Or it may be possible to have the server deliver a message to the user telling them where to install it, or helping them automatically do so.

When QS is installed, the install program will configure QS as being associated with the MIME type above. This will automatically ensure that IE 5+ associates the type with QS. The install program may also attempt to configure Netscape to treat QSB as a helper application for this MIME type, or identify and configure other browsers to properly view QSBdocuments.

If the user's browser is not configured to run QS, then the user may be asked if they want to save or open the file. Of course, the server will not deliver secure content in such a circumstance, since the secure browser is not operative to generate an anticipated response. Even if the user's computer system is configured to run QS, they may have to confirm that they want to. Provided the user's browser is set correctly, this will call QS to run the document.

QS then:

- Checks the checksum within the QMSB file is valid. If not, it refuses to run, and generates a message
- Checks the version within the QMSB file, and if the version is greater than supported by the QSB, refuses to run, saying that it needs to be upgraded
- Constructs a call to QSBcheck, using the URL in the file
- Includes the parameters
- Includes a checksum in the HTTP header for QSBcheck to check

Page 21, line 1:

Example 4

A Learning Management System (LMS) is a web-based software program that manages how learners access electronic and class-based learning. Commonly a participant logs into an LMS in a browser and then is able to take courses, assessments and be directed to appropriate places that allow learning and/or assessment to take place.

LMSs can call assessments using a variety of protocols including a standard promulgated by the AICC called AICC HACP, a specification promulgated by ADL SCORM called the SCORM 1.2 Runtime Environment and proprietary protocols. LMSs can make calls to Questionmark Perception via any of these means using a protocol called Perception Integration Protocol (PIP).

The use of PIP is controlled by an ASCII PIP file on the Perception Server, which defines what sort of interaction from the LMS is permitted. It's possible to define that Questionmark Secure (a secure browser) is popped up from the LMS by making this setting in a PIP file. For example the following PIP file makes all calls from an LMS via this PIP file call Questionmark Secure and it also arranges that the Home button at the end of assessment closes the secure browser.

```
; qsTest.pip
; demonstrates invoking Questionmark Secure
; sets Home button to close QS at end
; call with session.dll?call=qsTest&NAME=<your_name>
; September 2003

[Input]
NAME=NAME
GROUP="Testing"
DETAILS="Questionmark Secure"
; amend session to match assessment ID on your system
SESSION="7446569868587320"

[Settings]
UseNotify=no
Require QS=yes
UseHome=yes
Home=javascript:SB_ExitQS();
```

Using a PIP file like this, it means that users of corporate LMSs from companies like Saba, Plateau, Docent, Thinq, or academic course management systems like Blackboard and WebCT, can invoke a secure browser without making any change to the LMS. Providing they can call Perception via one of the supported protocols via PIP, then a secure browser can come up when the assessment is taken – the participant uses an ordinary browser to run the LMS and then a secure browser to take the assessment.

In view of this considerable relevant disclosure, which both provides antecedent basis in the specification as filed for the claim language, and illuminates and enables the practice of the claimed invention, it is therefore respectfully submitted that the application has ample support for the claim language, including a proper written description, to the extent that any such independent requirement exists under 35 U.S.C. § 112. The scope of the claims is also reasonably curtailed with respect to the scope of the disclosure.

Therefore, it is respectfully submitted that the rejections under 35 U.S.C. § 112 be reversed.

REJECTION OF CLAIMS 1-4 AND 6-20 UNDER 35 U.S.C. § 102(e).

Claims 1-4 and 6-20 are rejected under 35 U.S.C. § 102(e) as being anticipated by Winneg et al., US 7,069,586.

Winneg et al. is respectfully distinguished in at least that the secure application is invoked based on an <u>identification and login credentials of a user</u>, which have nothing whatsoever to do with any "received data encoding type" (claim 1) or "received data type encoding" (claim 9), and the secure application is used to retrieve the protected content <u>regardless of any such data encoding type or data type encoding</u>. Applicant respectfully submits that a "user type" is clearly distinct and non-overlapping with a "data encoding type" or a "data type encoding".

As is apparent, Winneg et al. defines the content and sources of information which are available only after login, and the determination of whether a secure "environment" is required is not based on any "received data encoding type" or "received data type encoding". Even if there are restrictions selective for particular exams, there is no teaching or suggestion that these restrictions are encoded in, or provided as part of the document requested, but rather these are provided as set-up information for the secure application.

Winneg et al. thus disclose that a secure application (which may include a secure browser-type application) is invoked, having predetermined restrictions, and only thereafter is a request for content issued from <u>within</u> the <u>already-invoked</u> secure application, which does not thereafter change in dependence on the content itself or a respective content encoding. Various users can have <u>different</u> usage restrictions with respect to the <u>same</u> content—some users are test creators, some are test graders, some are test takers. Since the same content is treated differently, it clearly cannot be the *content* itself which controls the restrictions, and therefore the present claims are distinguished from Winneg et al..

The security restrictions of Winneg et al. are based on the user login and the content *to be retrieved*, and such restrictions precede the content retrieval, and are thus not established after the content is retrieved based on its encoding. Likewise, since the secure environment is initiated before the request is made, and the secure environment does not normally terminate during use, insecure content cannot be employed within an insecure or normal browser. Thus, it is not disputed that Winneg et al. disclose a "secure" application. However, the disclosure of Winneg et al. is specific for an application which modifies the operating system environment, and Winneg et al. is not enabling in its description of how an invokable browser within that environment could perform the functions stated in the present claims, which require access to a normal or insecure browser based on a data type encoding or data encoding type. On the other hand, col. 16 of Winneg et al. precludes use of a normal browser, since this would permit the user to request uncontrolled content. This limitation arises, inter alia, because the secure application of Winneg et al. does **<u>not</u>** analyze a content type or content encoding.

Therefore, Winneg et al. employs a materially different method. For example, with respect to claim 1, the application of Winneg et al. which requests a document, does not perform the step of "automatically determining, based on a received data encoding type, whether a secure browser or a normal browser is to be employed, the secure browser having a set of functionality restricted with respect to the normal browser, to enhance security of a received document against data export". Likewise, with respect to claim 9, Winneg et al. does not perform the step of "automatically determining, based on a received data type encoding, whether a secure browser is required to be employed by a content provider or whether an insecure browser is to be employed, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser". The decision to use a secure application is made based on a user login, and independent of the "data encoding type" or "data type encoding".

Winneg et al. is thus respectfully distinguished in that a "professor" and a "student" can access the very same document (having the same data type encoding or data encoding type) and be afforded different security privileges based on their login; in accordance with the present claims, it is the ***data*** which determines the browser as being "normal" or "insecure" on one hand, or "secure" on the other, not the ***user identification***, which in turn defines the set of privileges available through the selected browser.

Winneg et al. provides a system in which a local software application controls the client computer, independent of a data type encoding or data encoding type. For example, Col. 6, lines 35-48 describe a system which defaults to a "secure" mode, and is machine status dependent, not received data encoding type dependent or data type encoding dependent. Indeed, the authorization to access or delete an exam is provided within the "secure" mode, and thus these functions are all provided within a single secure application. Therefore, the decisions 114, 116 do not serve to switch different "browsers" which are normal/insecure and secure. Col. 8, lines 48- Col. 9, line 44. Throughout the entire exam process, the machine is locked in a "secure" mode, maintaining this mode apparently independent of data and its associated type encoding or encoding type.

Thus, Winneg et al. teach a different system and method, with different results. Further, the system and method of Winneg et al. would be inoperative for its intended purpose if one sought to alter the access privileges to be dependent on a "type encoding" of a document, and therefore it cannot also be said that this is an obvious modification.

In formulating the rejection, the Examiner cites various portions of Winneg et al., which it is respectfully submitted do not teach or suggest at least the foregoing aspect of the claims. For example, the Examiner cites Col. 4, lines 3-5 for the proposition that Winneg et al. teach "automatically determining, based on a type encoding of the received data, whether a secure browser or a normal browser is to be employed". However, at this passage, Winneg et al. state: "The application being securely executed may be of any of a variety of types of applications, for example, a browser application or an application for receiving answers to questions of an examination (i.e., an exam taking application)." Thus, while Winneg et al. appear to disclose a secure browser <u>mode</u>, they utterly fail to disclose that a normal or insecure mode is also selectively available, in dependence on a data encoding type or data type encoding, having a different level of functionality.

The secure mode of Winneg et al. appears to be initiated based on a boot sequence, operating system limitation or user login. Col. 6, lines 35-67. Col. 9, lines 45-47, 50-55 and Col. 10, lines 10-13 indicate that a **<u>user input</u>** (and not a type encoding) determines which application to initiate. ("For example, FIG. 7 illustrates a GUI that may be displayed to a user to determine which application to initiate for the exam." "After the user has entered the class name and the professor in their respective fields and clicked on the OK button, the exam-taking application may use this information to determine a first application to be executed so that the student may take the exam (i.e., provide responses to one or more questions) and to determine the content (e.g., the questions of the exam or material to assist the user in taking the exam), if any, to be displayed by the first application." "Else, after hitting the 'OK' button of the GUI, next, in Act 122, secure execution of the exam-taking application may be initiated.").

The word "encoding" means, for example:

> The way in which symbols are mapped onto bytes, e.g. in the rendering of a particular font, or in the mapping from keyboard input into visual text; A conversion of plain text into a code or cypher form (for decoding by the recipient) **en.wiktionary.org/wiki/encoding**

While this definition is not per se binding on the Board, it is informative, and clearly inconsistent with the examiner's interpretation, which encompasses something different. In particular, a "user type" is clearly distinct and non-overlapping with a "data encoding type" or a "data type encoding".

This difference can be clearly seen in the passage at Col. 25, line 52-Col 26, line 10:

> The exam creation module may enable users (e.g., instructions or professors) to create an exam and an encrypt the exam for later use by an exam-taking application. The exam creation module may provide users one or more options for the format of an exam. The exam creation module may provide a first option of creating an exam that does not include any exam content (i.e., questions), which may be considered a "bluebook" type of exam. The exam creation module also may provide the option of creating an exam that includes exam content (i.e., questions). Before running the exam creation module, a user (e.g., a professor, instructor or teaching assistant) may create an exam content file by using a word processing application such as Microsoft Word. The exam creation module then may be used to provide a password for and/or encrypt the exam content file.
>
> In a first act, the user may be prompted to enter the user's ID and password.
>
> In a next act, the exam creation module may enable the user to locate the exam content file, for example, by enabling use of a browser application.
>
> In a next act, the user may be prompted to enter a password that any students who will take the exam will be required to enter in order to take the exam and access the exam content file. The password and exam content then may be saved together in the exam content file.

As will be apparent, the Winneg et al. application and method defines the content and sources of information which are available only after login, and it is not any "received data encoding type" or "received data type encoding" which controls restrictions on functionality.

Even if there are restrictions selective for particular exams, there is no teaching or suggestion that these restrictions are encoded in or as part of the document requested, but rather these are provided as set-up information for the secure application (Col. 17):

> Returning to method 140, in a next Act 144, any instances of the first application currently executing on the computer system may be terminated. As described above, the first application should be included on a list of authorized processes, else the first application would be terminated by performance of Acts 302 308 described below. However, prior to performance of method 300, one or more instances of the first application already may be executing on the computer system, and may be accessing unauthorized content and/or executing unauthorized processes.
>
> Accordingly, prior to performance of Act 146, any instances of the first application currently executing on the computer system may be terminated. To determine if an instance of the first application is currently executing on the computer system, a list of processes currently executing on the computer system may be accessed. The list may contain one or more entries, where each entry contains an identifier of a process currently executing on the computer system. Each entry may be accessed to determine whether the identifier of the entry is an identifier for the first application. If a match is found, the instance of the first application may be terminated and the identification of the instance may be removed from the list of currently executing processes.

Winneg et al. thus disclose that a secure application (which may include a secure browser) is invoked, having predetermined restrictions, and *then* a request for content is issued, within the secure application, which does not change in dependence on the content itself or a respective content encoding. Different users can have different usage restrictions with respect to the same content—some users are test creators, some are test graders, some are test takers. Since the same content is treated differently, it clearly cannot be the *content* which controls the restrictions.

These restrictions are therefore based on the user login and the content *to be retrieved*, and such restrictions precede the content retrieval, are not established after the content is retrieved based on its encoding.

It is not disputed that Winneg et al. disclose a secure application. However, the disclosure of Winneg et al. is somewhat specific for an application which modifies the operating system environment, and Winneg et al. is not enabling in its description of how an invokable browser could perform the stated functions. Indeed, column 16 seems to indicate that the secure application is itself **not** a browser, stating:

> Further, the first application may be configured such that hyperlink functionality is not available within the first application. Thus, a user cannot type in a uniform resource locator (URL) and automatically launch a browser application that hyperlinks the user outside of the first application.

Likewise, Col. 19 states:

> If Act 306 includes both looping through an authorized process list and looping through an unauthorized process list, the unauthorized process list may function as an added security measure. The unauthorized process list may include one or more processes that a controlling entity is particularly concerned about executing during execution of the first application. For example, the unauthorized process list may include browser applications, (e.g., Microsoft Internet Explorer), applications for scheduling tasks to be performed on the computer system, (e.g., Microsoft Task Scheduler), and applications for managing tasks performed on the computer system, (e.g., Microsoft Task Manager). Terminating task-managing manager and task-scheduling applications prevents a process (e.g., an application) that has been scheduled to execute during execution of the first application from executing.

Further supporting disclosure in Winneg et al. is as follows:

Col. 6:

> DESCRIPTION
>
> An illustrative embodiment of a method of securely executing an application such that unauthorized content is prohibited from being either accessed or viewed by a user of the computer system during execution of the application is described below in the context of executing an exam-taking application. Such an illustrative embodiment is not meant to limit the scope of any of the claims set forth below and is provided merely for illustrative purposes, as such method may be used in a variety of other contexts, for example, in the context of executing a browser application.

As used herein, an "exam-taking application" is an application configured to enable a user to use a computer system to provide one or more answers to one or more questions of an exam.

*   *   *

In Act 102 it may be determined whether execution of this exam-management application is the result of the computer system being rebooted during an exam-taking application.

For example, as described below in relation to Act 159, the computer system may be configured such that, if the computer system is rebooted, execution of the exam-taking application is re-initiated. For example, in Act 159 of FIG. 8, a computer system parameter (i.e., flag) may be changed from its default value such that upon booting the computer system, the student exam-management application is automatically initiated....

*   *   *

Accordingly, as a result of the computer system being booted, such key may be accessed, to determine if the student exam-management application should be initiated.

Further, as described below in relation to Act 224 of FIG. 14, as a result of exiting the exam-taking application, the computer system parameter (e.g., auto-run key) may be set back to its original default value so that the exam management application is not initiated upon reboot.

Thus, Act 102 may include ascertaining whether this auto-run key is set to a logical value indicating that execution of the exam-management application is the result of the computer system being rebooted during an exam-taking application.

Col. 8:

If, in Act 108, it is determined that the login is successful because the entered student ID and the entered student password are both valid, then, in Act 112, the user may be enabled to select an action from two or more actions corresponding to exams. For example, Act 112 may include providing a GUI that displays a menu from which a user may select an action from two or more actions corresponding to exams.

For example, Act 112 may include providing the GUI illustrated in FIG. 4. This GUI provides a user with the option of taking an exam, deleting prior exams or copying a prior exam to a diskette.

In a next Act 114, it may be determined whether the user selected to copy an exam file, for example, by selecting the appropriate radio button from the GUI of FIG. 4.

If the user selected to copy an exam file in Act 114, then, in Act 116, the user may be enabled to copy an exam file.

Act 116 may include enabling a student to copy an exam from an exam directory to another location, for example, a floppy disk or another network address. This enables the student to make copies of an examination that can be attached to an email, stored as a backup or sent to a network directory corresponding to a course for which the exam was taken. As described below, upon completion of an exam, an exam file containing the student's responses to the exam questions may be stored in a predefined directory. Accordingly, Act 116 may include accessing such predefined directory and selecting an exam to copy to another location.

Col. 10

FIG. 8 is a flowchart illustrating an example of a method 130 securely executing an exam-taking application such that a user is prohibited from accessing or viewing unauthorized content during execution of the exam-taking application.

In Act 141, any capabilities provided by the computer system by which a user of the computer system may access or view unauthorized content may be disabled. Disabling such capabilities may be accomplished using any of a variety of techniques.

Col. 13:

Accordingly, Act 210 may include providing a dynamic link library (DLL) that includes a plurality of hooks for intercepting events (e.g., messages, mouse action, key strokes) before they reach an application, for example, the first application described in more detail below. Further, one or more of the hooks of the DLL may be configured to prevent the event from reaching the first application, before performing an action before sending the event to the application.

Thus, Act 210 may include installing such a DLL on the computer system such that one or more of the hooks may be utilized during execution of the exam-taking application.

Appendix III is an example of a DLL, Hooksdll.C written in the C programming language that may be installed as part of Act 210 and whose hooks may be utilized during execution of the first application described below. Other processes written in other programming languages or a combination of one or more programming languages, including C, may be used as part of Act 210 to implement programming hooks that prevent unauthorized content from being accessed by or displayed by the first application.

The DLL for providing the hooks (hereinafter Hooksdll) may include a function to assign a value to a variable of the computer system, where this variable may be accessed by other functions to determine that HooksDll has been installed and initialized.

Col. 14:

Accordingly, Hooksdll also may include a function to intercept a message sent from the OS to the first application that responds to a user hitting the "Ctrl" and the "T" keys concurrently, and sending a message to the first application to display the timer.

Hooksdll also may include a function to intercept all messages sent from the OS to a destination window (e.g., a window of first application) in response to a user hitting a key or combination of keys of the keyboard. A return value for this hook may be set to a non-zero value to disable certain key combinations. For example, this hook may be used to disable function keys F1 F10, to disable ALT_SHIFT DLL, any of the hook keys listed below in Table 1, or any of a variety of other keys or key combinations.

Hooksdll also may include a hook to intercept any messages from the operating system to a destination window (e.g., a window of the first application) in response to a mouse event, for example, double clicking the left mouse button, right clicking the right mouse button, or moving the scroll wheel of the mouse. This hook may be configured to return a non-zero value such that the mouse message is not sent to the destination window. Further, the hook may be configured to determine the mouse position and, based on the mouse position, disable one or more mouse events. For example, this hook may be used to disable one or more of the mouse events listed below in Table 2.

Col. 22:

In a next Act 158, the exam file may be opened by the first application so that the exam may begin. In other words, the student may be enabled to enter one or more responses to one or more questions of the examination.

* * *

In a following Act 182, the one or more unauthorized processes terminated before execution of the first application may be initiated. For example, the list of terminated processes described above in relation to Act 208 may be accessed and each process in the list may be initiated. Thus, if a user had several applications, for example, an email application and a browser application executing before execution of the exam-taking application began, these applications will be initiated so that the user's computer system is returned to the state that it was in before the user executed the exam-taking application.

Therefore, it is clear that Winneg et al. employ a materially different method. For example, with respect of claim 1, the application of Winneg et al. which requests a document does not perform the step of "automatically determining, based on a received data encoding type, whether a secure browser or a normal browser is to be employed, the secure browser having a set of functionality restricted with respect to the normal browser, to enhance security of a received document against data export". Likewise, Winneg et al. does not perform, per claim 9, the step of "automatically determining, based on a received data type encoding, whether a secure browser is required to be employed by a content provider or whether an insecure browser is to be employed, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser". This decision is made based on a user login, and independent of the "data encoding type" or "data type encoding".

A particularized analysis per claim group follows.

CLAIMS 1-4 and 6-8

The examiner interprets the phrase "type encoding" in claim 1 to encompass a login classification of the user. This, however, is an erroneous interpretation of the claim.

The complete claim phrase of claim 1 is "...based on a type encoding of the received data..." Therefore, this language does not relate to a type of **USER**, but rather a type of **DATA**. Likewise, the result of this "type encoding" in accordance with the claims is the selection of a secure browser or a normal browser, each with a respectively different level of functionality, and not a set of privileges of the user within a singular browser type.

It is especially noted that, in accordance with claim 1, the decision of whether to employ a secure browser or a normal browser is automatically determined based on a data encoding type of the received data. Therefore, it is not directly the server, but the respective client, which automatically determines which browser to employ, and that this determination is automatically made based on a type encoding of the data received.

According to Winneg et al., Col. 9, lines 59-67 provide that it is the information entered in the fields of Fig. 7 that are used to determine if content is to be displayed by "the first application" (e.g., MS Word). Fig. 7 shows a login screen, in which a user enters class name, professor and exam date. This does not correspond to the document requested by the browser from the cooperative server, and received by the browser in response to the request, as provided by claim 1.

Therefore, it is seen that Winneg et al. employ a presumption that, so long as the exam-taking application is engaged, the machine <u>must</u> be in the "secure" mode, and do not employ encoding of requested data received from the server to automatically control whether a secure browser or insecure browser is employed. This differs from the present claim 1, which permits, for example, the server to dynamically serve content with different data encoding type to control the browser, and therefore expressly permit use of an insecure browser, for example where a requested task permits or compels such an environment.

The dependent claims 2-8 are believed to be distinguished at least on the same basis as claim 1.

CLAIMS 9-14 and 17-20

The examiner interprets the phrase "type encoding" in claim 9 to encompass a login classification of the user. This, however, is an erroneous interpretation of the claim. The complete claim phrase of claim 9 is "automatically determining, based on a received data type encoding, whether a secure browser is required to be employed by a content provider or whether an insecure browser is to be employed, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser".

Therefore, this language does not relate to a type of **USER**, but rather a type of **DATA**. Likewise, the result of this type encoding in accordance with the claims is the selection of a secure browser or a normal browser with respectively different level of functionality, and not a set of privileges of the user within a singular browser type.

It is especially noted that, in accordance with claim 9, the decision of whether to employ a secure browser or an insecure browser is automatically determined based on a data type encoding of the received data. Therefore, it is not directly the server, but the respective client, which automatically determines which browser to employ, and that this determination is automatically made based on a data type encoding of the data received by the browser.

Therefore, it is seen that Winneg et al. employ a presumption that, so long as the exam-taking application is engaged, the machine must be in the "secure" mode, and do not employ encoding of requested data received from the server to automatically control whether a secure browser or insecure browser is employed. This differs from the present invention in accordance with claim 9, which permits, for example, the server, by serving content with different type encoding, to dynamically control the browser based on data encoding.

Claim 9 is particularly distinguished in that Winneg et al. employ only a single browser type, and not both a separately defined secure browser and an insecure browser, the use of which is determined automatically by a content provider. As discussed above, the decision by Winneg et al. of whether to employ security or not is made in dependence on a login status, and therefore, Winneg et al. do not teach or suggest at least "automatically determining <u>whether a secure browser is required to be employed by a content provider</u> or whether an insecure browser is to be employed, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser," as provided in claim 9.

The dependent claims 10-20 are believed to be distinguished at least on the same basis as claim 9.

CLAIM 15

It is particularly noted that claim 15 makes the distinctions more explicit, and this claim is therefore argued separately.

Claims 15 provides:

> 15. The user interface method according to claim 9, wherein the secure browser is initiated based on a type encoding of the received data.

Thus, claim 15 makes clear that "the secure browser is initiated based on a type encoding of the received data," which is the same data that was requested, and is this clearly not open to an interpretation that some different data, such as the user login information which is provided FROM the browser TO the cooperative server, is the basis for the selection of the secure browser.

CLAIM 16

It is particularly noted that claim 16 also makes the distinctions more explicit, and this claim is therefore argued separately.

Claims 16 provides:

> 16. The user interface method according to claim 9, wherein the secure browser is initiated based on a code associated with the secure content.

Claim 16 requires that "the secure browser is initiated based on a code associated with the secure content" Therefore, it is clear that this distinguishes the secure application of Winneg et al., which is initiated <u>before</u> any content (secure or otherwise) is requested, and thus could not be initiated <u>based on</u> a code associated with the secure content.

REJECTION OF CLAIMS 5 AND 21 UNDER 35 U.S.C. § 103(a)

Claims 5 and 21 are rejected under 35 U.S.C. § 103(a) as being obvious over Winneg et al. US 7,069,586 in view of Chang US 2002/0097416.

In a typical computer system, it is not the browser which converts "text information" to "graphic objects"; rather, there is a display driver system or operating system separate from the application, which receives the source information to be displayed, and then formats it for presentation. Therefore, similar to the architecture of Chang et al., the rasterizer services a plurality of applications, and is separate therefrom.

CLAIM 5.

Claim 5 is distinguished at least on the same basis as claim 1.

In addition, it is noted that claim 5 provides a different configuration from Chang et al., in which the rasterizer (conversion from text information to a graphic object) is part of the secure browser process, i.e., "the secure browser renders text information as graphic objects." Therefore, Winneg et al., US 7,069,586 and Chang et al., US 2002/0097416 are believed distinguished. In particular, it is not believed that Chang et al. disclose an application-level rasterizer, but rather a rasterizer that services all applications on a device, and therefore is distinguished by claim 5 which requires that the secure browser, an application, perform the conversion.

CLAIM 21.

Claim 21 is distinguished at least on the same basis as claim 9.

In addition, it is noted that claim 21 provides a different configuration from Chang et al., in which the rasterizer (conversion from text information to a graphic object) is part of the secure browser process, i.e., "the secure browser renders text information as graphic objects." Therefore, Winneg et al., US 7,069,586 and Chang et al., US 2002/0097416 are believed distinguished. In particular, it is not believed that Chang et al. disclose an application-level rasterizer, but rather a rasterizer that services all applications on a device, and therefore is distinguished by claim 21 which requires that the secure browser, an application, perform the conversion.

Therefore, it is respectfully requested that the rejections be reversed.

Respectfully submitted,

[illegible]

/Steven M. Hoffberg/
Steven M. Hoffberg
Reg. No. 33,511

HOFFBERG & ASSOCIATES
10 Bank Street-Suite 460
White Plains, NY 10606
(914) 949-2300

(viii) Claims appendix.

1. A secure user interface method, for interacting with a user through a browser, comprising:

controlling the browser to request a document from a cooperative server, the browser providing data export support functionality;

receiving data with the browser in response to the request;

automatically determining, based on a received data encoding type, whether a secure browser or a normal browser is to be employed, the secure browser having a set of functionality restricted with respect to the normal browser, to enhance security of a received document against data export;

receiving the secure content for presentation in the secure browser; and

communicating an input from the user, through the secure browser, to a cooperative server.

2. The user interface method according to claim 1, further comprising the step of limiting access of a user, with the secure browser, to documents outside of a specified set.

3. The user interface method according to claim 1, further comprising the step of authenticating the secure browser, to assure that the secure browser having the restricted set of functionality is available for presentation of secure content.

4. The user interface method according to claim 1, wherein the secure browser lacks one or more of the following functions: print, save, cache, cut and copy.

5. The user interface method according to claim 1, wherein the secure browser renders text information as graphic objects.

6. The user interface method according to claim 1, wherein the secure browser restricts termination of execution of the secure browser.

7. The user interface method according to claim 3, wherein the secure browser restricts termination of execution of the secure browser.

8. A computer readable media storing a program for a general purpose computer for performing the method of claim 3

9. A secure user interface method, for interacting with a user through a browser, the browser providing a set of navigational functionality, comprising:

requesting a document from a cooperative server;

receiving data in response to the request;

automatically determining, based on a received data type encoding, whether a secure browser is required to be employed by a content provider or whether an insecure browser is to be employed, the secure browser restricting interaction of the user with tasks other than those permitted by the secure browser which are permitted by the insecure browser;

invoking the secure browser;

receiving the secure content for presentation in the secure browser; and

communicating an input from the user, through the secure browser, to a cooperative server.

10. The method according to claim 9, wherein the secure browser provides restricted navigational functionality with respect to the navigational functionality of the insecure browser alone.

11. The user interface method according to claim 9, further comprising the step of limiting access of a user, with the secure browser, to access of documents within a specified set.

12. The user interface method according to claim 9, further comprising the step of authenticating the secure browser at a remote server prior to presenting the secure content to ensure that the content will only be delivered in the secure browser.

13. The user interface method according to claim 9, wherein the secure browser prevents use of the following functions: save, copy, and navigate to unrestricted documents.

14. The user interface method according to claim 9, wherein the secure browser restricts termination of execution of the secure browser.

15. The user interface method according to claim 9, wherein the secure browser is initiated based on a type encoding of the received data.

16. The user interface method according to claim 9, wherein the secure browser is initiated based on a code associated with the secure content.

17. The user interface method according to claim 9, wherein the secure browser is granted principal application level control over graphic user interface inputs from a user.

18. The user interface method according to claim 9, wherein the secure browser is granted exclusive control over graphic user interface functionality when invoked.

19. The user interface method according to claim 9, further comprising the step of authenticating the server by the secure browser prior to presenting the secure content.

20. A computer readable media storing a program for a general purpose computer for performing the method of claim 9.

21. The user interface method according to claim 9, wherein the secure browser renders text information as graphic objects.

(ix) Evidence appendix.

Not Applicable.

(x) Related proceedings appendix.

Not Applicable.